Multi-Masks: a Set-Centric Mask Output for Production Rendering

Paolo Berto Durante* — J Cube Inc. / Polygon Pictures Inc.
Aghiles Kheffache† — The 3Delight Team

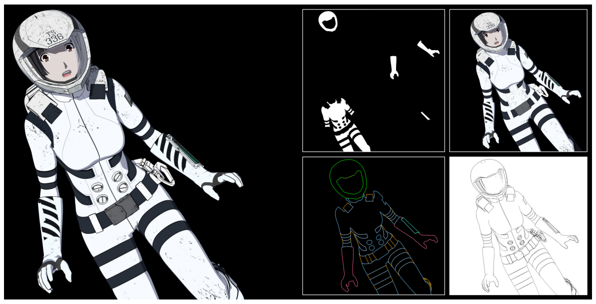

Figure 1: *Concurrent "multi-masks" output and edge detection. Knights of Sidonia ©T. Nihei · Kodansha / KOS Prod. Committee*

1 Abstract

We introduce a user-friendly workflow (which we call "multi-masks") to define, manipulate and *concurrently* render coverage masks. We show the benefits of a *set-centric* approach, lay the basis for a successful implementation from a user interface perspective and explain the rendering technology requirements. We show how multi-masks can be used to perform edge detection as a mean of producing "inking" effects.

2 Overview

Masks are bitmaps (similar to *mattes*) and are convenient to separate and manipulate specific scene elements during compositing or color grading [Fielding 1980]. For example, one can separate foreground and background elements (or elements with specific material assignment) to perform a color correction.

A common practice in production is to manually assign numerical ID's to scene elements or to use custom material output colors[1] to get masks out of the renderer and into the compositing application. This approach involves a tedious manual process, requires many sequential renders and is generally error prone. In addition, effects such as depth of field and motion blur cause problems due to difficulties of filtering arbitrary integer values. Lastly, artists do not have a *global view* of how masks are setup and assigned.

Another approach is to modify shaders and add custom AOVs to output specific masks. This approach is even more involved and necessitates additional technical resources.

The "multi-masks" workflow introduced here is both user friendly and flexible, allowing output of coverage masks for *arbitrary* elements (objects, materials or a hybrid of both) while requiring only simple additions to the DCC application and the renderer.

3 Implementation

Concepts We use *Maya* sets[2] to *define* masks (we call these sets "mask sets"). Sets' given *names* are used throughout the application as unique identifiers. Scene elements, either geometry or materials or both, are then added to these sets to define the visual "contour" of the coverage masks. This allows artists to easily create arbitrary masks at an early stage in the pipeline (such as during modeling or look development) without introducing arbitrary quantities such as manually assigned IDs.

Secondly, we use automatically generated *layers* for each mask set. These layers can then be specified to the renderer as images to output. Any quantity of such layers and their corresponding mask sets can be defined.

User Interface The application needs a UI to handle addition, removal and listing of set elements. This is readily available in *Maya* and most DCC applications through set manipulation tools. Secondly, the rendering plug-in needs to provide an interface to access these sets through named layers. This interface is akin to AOV selection UI so layers can simply be added to the list of available AOVs (or any other UI-equivalent).

Rendering Core Requirements The renderer must be able to output multiple layers in parallel[3] with potentially *different visibility functions for each layer*. Naturally, the set of visible elements is provided for each layer to the renderer. We implemented this workflow in *3Delight*, supporting both REYES and the ray tracing visibility algorithms. If desired, we use the flexibility of the *OpenEXR* file format [Kainz et al. 2006] to store all masks into one file.

4 Results

We implemented multi-masks at *Polygon Pictures Inc.* and greatly improved the color grading workflow. Interestingly, the generality of our approach allowed us to solve a frequent hurdle in *anime*-style renderings: generation of edge contours (or *outlines*) on object features that are "orthogonal" to assets' hierarchy or shading setup. As shown in Figure 1, we used multi-masks to define the atlas of all needed outlines and then use *3Delight*'s sub-sample edge detection feature to generate contours on the corresponding mask layers.

5 Conclusion and Future Work

We laid the basics of a user friendly coverage masks system that does not rely on numerical IDs or similar *ad-hoc* techniques. As a bonus to the more general use of coverage masks for grading, we were able to use the multi-masks system to solve common problems with outline generation. An important improvement to this system is a more refined definition of "mask regions", for example, a texture could be used to define a function on top of our elements-based mask description. The semantics and the user interface of this functionality are the premises of ongoing work.

References

FIELDING, R. 1980. A technological history of motion pictures and television. 146–.

KAINZ, F., BOGART, R., AND STANCZYK, P. 2006. Technical introduction to openexr. Industrial Light & Magic.

*paolo@j-cube.jp
†aghiles@3delight.com

[1]Sometimes encoding one different mask in each RGB channel.
[2]Equivalents are trivially implemented in other DCC Apps.
[3]Sequential output is obviously possible but defeats the purpose.